ELSEVIER Journal of Computational and Applied Mathematics 65 (1995) 369–385

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Gaussian quadrature for matrix valued functions on the real line

Ann Sinap*

Department of Mathematics, Katholieke Universiteit Leuven, Celestijnenlaan 200 B, B-3001 Heverlee, Belgium

Received 4 November 1994; revised 9 March 1995

Abstract

Parallel versions of the block Chebyshev algorithm to generate the recursion coefficients of orthonormal matrix polynomials and the Gaussian quadrature algorithm to approximate matrix integrals on the real line are implemented on an SP1.

Keywords: Orthogonal matrix polynomials; Block tridiagonal matrices; Quadrature; Parallel algorithms

AMS Classification: 42C05; 65D32; 41A55; 47A56; 65Y05

1. Introduction

The aim of this paper is to implement the Gaussian quadrature rules for matrix-valued functions on a multiprocessor. In [26] we have constructed quadrature formulas, using orthonormal matrix polynomials, to approximate matrix integrals on the real line. Now we will give a formula to determine the quadrature weights using the eigenvalues and first p components of the normalized eigenvectors of a symmetric block tridiagonal matrix. Since the elements of the main and first subdiagonal are the recursion coefficients of the corresponding orthonormal matrix polynomials, the block version of the modified Chebyshev algorithm to compute these recursion coefficients from the modified moments is constructed.

The algorithms are implemented on a multiprocessor based on RiSC-processors. Some RiSC-processors, with their own memory and disk, are connected and put together in one box. This leads to a multiprocessor with distributed memory. Different manufacturers use this principle, for example IBM has put together some of their RiSC-processors and calls the new configuration Scalable Power machine.

To take care of the communication between different processors, the user needs instructions which he can call from his C or Fortran programs. These instructions are part of a message passing

* E-mail: ann.sinap@wis.kuleuven.ac.be.

SSDI 0377-0427(95)00124-7

library. Linda, PVM, MPL, Express,... are examples of such libraries. We use PVM, Parallel Virtual Machine, which is developed at the Oak Ridge National Laboratory and is publicly available.

In Section 2 we explain what matrix polynomials are, introduce orthogonal matrix polynomials on the real line and discuss some properties which are needed in the following sections. In Section 3 we give the Gaussian quadrature rules and in Section 4 we discuss two parallel algorithms. The first one is the block version of the modified Chebyshev algorithm and the second one is an implementation of the Gaussian quadrature rules.

2. Orthogonal matrix polynomials

If $A_0, A_1, \ldots, A_n$ are elements of $\mathbb{R}^{p\times p}$, $A_n \neq 0$ and $x \in \mathbb{R}$, then we call

$$P(x) = A_n x^n + A_{n-1}x^{n-1} + \cdots + A_1 x + A_0,$$

a *matrix polynomial* of degree n. A point x_0 is a *zero* of P if $\det P(x_0) = 0$. Note that if the leading coefficient of P is nonsingular, $\det P(x)$ is a polynomial of degree np. Other important notions can be found in [18, 26, 27].

Let $\mathbb{R}^{p\times p}[x]$ be the set of polynomials in a real variable x whose coefficients are $p \times p$ matrices with real entries and let W be a weight matrix function integrable over $[a, b]$ (see [21, 17]), i.e. $W(x)$ is a nonnegative definite matrix for every $x \in [a, b]$. We can define two inner products on $\mathbb{R}^{p\times p}[x]$: the left inner product is given by

$$\langle P, Q\rangle_{\mathrm{L}} = \int_a^b P(x)W(x)Q(x)^{\mathrm{T}}\,\mathrm{d}x$$

and the right inner product satisfies

$$\langle P, Q\rangle_{\mathrm{R}} = \int_a^b P(x)^{\mathrm{T}}W(x)Q(x)\,\mathrm{d}x,$$

where $P, Q \in \mathbb{R}^{p\times p}[x]$.

A generalization of the Gram–Schmidt orthonormalization procedure for the set $\{I, xI, x^2I, \ldots\}$ with respect to these inner products produces a set of left orthonormal matrix polynomials $\{P_n\}_{n=0}^{\infty}$ which satisfy

$$\langle P_n, P_m\rangle_{\mathrm{L}} = \int_a^b P_n(x)W(x)P_m(x)^{\mathrm{T}}\,\mathrm{d}x = \delta_{n,m}I$$

and a set of right orthonormal matrix polynomials $\{Q_n\}_{n=0}^{\infty}$ satisfying

$$\langle Q_n, Q_m\rangle_{\mathrm{R}} = \int_a^b Q_n(x)^{\mathrm{T}}W(x)Q_m(x)\,\mathrm{d}x = \delta_{n,m}I.$$

Moreover, P_n and Q_n are matrix polynomials of degree n, with nonsingular leading coefficient. The left orthonormal matrix polynomials P_n are defined up to a multiplication on the left by an

orthogonal matrix and the right orthonormal matrix polynomials Q_n are defined up to a multiplication on the right by an orthogonal matrix.

Since the left and right orthogonal polynomials are closely related, $Q_n = P_n^T$, we restrict ourselves to the discussion of the left inner product $\langle \cdot, \cdot \rangle_L$.

As in the scalar case, the orthonormal matrix polynomials are orthogonal to every matrix polynomial of lower degree and they satisfy a three term recurrence relation (see [21, 17])

$$xP_n(x) = D_{n+1}P_{n+1}(x) + E_nP_n(x) + D_n^TP_{n-1}(x), \quad n \geqslant 0,$$

where $P_{-1} = 0$, $P_0 = I$, D_n is nonsingular and E_n is symmetric.

Using this recurrence relation we can show that the zeros of P_n can be determined as the eigenvalues of a symmetric block tridiagonal $np \times np$ matrix.

Theorem 2.1. *Let x_0 be a zero of P_n with associated right root vector v_0, then x_0 is an eigenvalue and the np-dimensional vector*

$$col(P_i(x_0)v_0)_{i=0}^{n-1}$$

is an eigenvector of the symmetric block tridiagonal matrix

$$J_n = \begin{pmatrix} E_0 & D_1 & & & & \\ D_1^T & E_1 & D_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & D_{n-2}^T & E_{n-2} & D_{n-1} \\ & & & & D_{n-1}^T & E_{n-1} \end{pmatrix}$$

Proof. Since x_0 is a zero of P_n with corresponding root vector v_0, the recurrence relation becomes

$$x_0P_0(x_0)v_0 = D_1P_1(x_0)v_0 + E_0P_0(x_0)v_0,$$

$$x_0P_k(x_0)v_0 = D_{k+1}P_{k+1}(x_0)v_0 + E_kP_k(x_0)v_0 + D_k^TP_{k-1}(x_0)v_0, \quad 1 \leqslant k \leqslant n-2,$$

$$x_0P_{n-1}(x_0)v_0 = E_{n-1}P_{n-1}(x_0)v_0 + D_{n-1}^TP_{n-2}(x_0)v_0.$$

The proof follows from putting these equations in matrix form. □

3. Gaussian quadrature on the real line

In [26] we have shown how an integral of matrix functions can be approximated by a sum, using orthonormal matrix polynomials.

Theorem 3.1. *Let (X, J) be a Jordan pair of the orthonormal matrix polynomial $P_n(x)$ on the interval $[a, b]$ and with respect to the weight matrix function $W(x)$. Then we have*

$$\int_a^b F(x)W(x)G(x)^T\,dx \simeq \sum_{i=1}^{k} F(x_i)\Lambda_iG(x_i)^T,$$

where k is the number of different zeros x_i, m_i is the multiplicity of x_i, $v_{i,j}$ are the vectors associated with x_i,

$$\Lambda_i = (v_{i,1} \ \dots \ v_{i,m_i}) L_i^{-1} \begin{pmatrix} v_{i,1}^{\mathrm{T}} \\ \vdots \\ v_{i,m_i}^{\mathrm{T}} \end{pmatrix}$$

and

$$L_i = \begin{pmatrix} v_{i,1}^{\mathrm{T}} \\ \vdots \\ v_{i,m_i}^{\mathrm{T}} \end{pmatrix} \sum_{j=0}^{n-1} P_j(x_i)^{\mathrm{T}} P_j(x_i) (v_{i,1} \ \cdots \ v_{i,m_i}).$$

This quadrature formula is exact for matrix polynomials $F(x)$ and $G(x)$ which satisfy

$$\deg F(x) + \deg G(x) \leqslant 2n - 1.$$

The formula to compute the quadrature weights Λ_i, $i = 1, 2, \dots, k$, looks very difficult but some straightforward matrix computations and Theorem 2.1 yields the following result.

Theorem 3.2. *Let $U^{(i,j)}$, $j = 1, 2, \dots, m_i$ be the eigenvectors of the matrix J_n associated with the eigenvalue x_i. Then the quadrature weights are given by*

$$\Lambda_i = (U_0^{(i,1)} \ U_0^{(i,2)} \ \dots \ U_0^{(i,m_i)}) G_i^{-1} \begin{pmatrix} U_0^{(i,1)\mathrm{T}} \\ U_0^{(i,2)\mathrm{T}} \\ \vdots \\ U_0^{(i,m_i)\mathrm{T}} \end{pmatrix},$$

where

$$(G_i)_{s,t} = U^{(i,s)\mathrm{T}} U^{(i,t)}$$

and $U_0^{(i,j)}$ is the vector consisting of the first p components of $U^{(i,j)}$.

Proof. Let x_i be an eigenvalue of J_n with multiplicity m_i and let $U^{(i,j)}$, $j = 1, \dots, m_i$, be the corresponding eigenvectors. Theorem 2.1 shows that the eigenvectors satisfy

$$U^{(i,j)} = \begin{pmatrix} U_0^{(i,j)} \\ U_1^{(i,j)} \\ \vdots \\ U_{n-1}^{(i,j)} \end{pmatrix} = \begin{pmatrix} P_0(x_i) v_{i,j} \\ P_1(x_i) v_{i,j} \\ \vdots \\ P_{n-1}(x_i) v_{i,j} \end{pmatrix} = \begin{pmatrix} v_{i,j} \\ P_1(x_i) v_{i,j} \\ \vdots \\ P_{n-1}(x_i) v_{i,j} \end{pmatrix},$$

such that $U^{(i,s)\mathrm{T}} U^{(i,t)} = v_{i,s}^{\mathrm{T}} \sum_{j=0}^{n-1} P_j(x_i)^{\mathrm{T}} P_j(x_i) v_{i,t} = (G_i)_{s,t}$. □

Further simplification occurs when the eigenvectors are normalized:

$$\Lambda_i = (V_0^{(i,1)}\, V_0^{(i,2)} \ \dots \ V_0^{(i,m_i)}) \begin{pmatrix} V_0^{(i,1)\mathrm{T}} \\ V_0^{(i,2)\mathrm{T}} \\ \vdots \\ V_0^{(i,m_i)\mathrm{T}} \end{pmatrix},$$

where $V_0^{(i,j)}$ is the vector consisting of the first p components of the normalized eigenvector $V^{(i,j)}$.

So we need to know the eigenvalues and the first p components of the normalized eigenvectors of a symmetric block tridiagonal matrix. The elements of this matrix are the recursion coefficients of the corresponding orthonormal matrix polynomials. Since these matrix polynomials are defined up to a multiplication on the left by an orthogonal matrix, we can choose this factor such that the recursion coefficients D_n are lower triangular matrices. In this case J_n becomes a symmetric bandmatrix with semi-bandwidth p.

4. Algorithms to approximate a matrix integral

4.1. Generation of the recursion coefficients

The recursion coefficients E_i and D_i, $i = 0, 1, \dots, n-1$, can be computed from the first $2n$ moments $S_k = \int_a^b x^k W(x)\,\mathrm{d}x$, $k = 0, 1, \dots, 2n-1$, by means of the block version of the Chebyshev algorithm. In [14–16] the Chebyshev algorithm is described for the classical case $p = 1$. Since for $p = 1$ it is better to use the modified Chebyshev algorithm, which computes the recursion coefficients from modified moments, we will treat the block version of this modified Chebyshev algorithm. A comparison of the conditioning of the modified Chebyshev algorithm versus the Chebyshev algorithm for power moments as done by Gautschi [14–16] is of interest but due to space limitations this will not be considered here.

Consider a set of matrix polynomials $\{Q_n\}_{n=0}^{\infty}$ which satisfy the recurrence relation

$$xQ_k(x) = A_{k+1}Q_{k+1}(x) + B_kQ_k(x) + C_kQ_{k-1}(x), \quad k = 0, 1, 2, \dots,$$

$Q_{-1} = 0$, $Q_0 = I$ and with $A_1, A_2, \dots$ nonsingular matrices.

The modified moments of the matrix weight function W with respect to $\{Q_k\}$ are given by

$$M_j = \int_a^b W(x)Q_j(x)^{\mathrm{T}}\,\mathrm{d}x = \langle I, Q_j \rangle.$$

Define

$$\sigma_{k,l} = \int_a^b P_k(x)W(x)Q_l(x)^{\mathrm{T}}\,\mathrm{d}x = \langle P_k, Q_l \rangle,$$

then $\sigma_{-1,l} = 0$, $\sigma_{0,l} = M_l$ and from the recurrence relations we see that the matrices $\sigma_{k,l}$ can be computed recursively by

$$D_k\sigma_{k,l} = \sigma_{k-1,l+1}A_{l+1}^{\mathrm{T}} + \sigma_{k-1,i}B_l^{\mathrm{T}} - E_{k-1}\sigma_{k-1,l} + \sigma_{k-1,l-1}C_l^{\mathrm{T}} - D_{k-1}^{\mathrm{T}}\sigma_{k-2,l}.$$

Let $L_{j,j}$ be the leading coefficient of Q_j, then $L_{j,j} = A_j^{-1}A_{j-1}^{-1}, \ldots, A_1^{-1}$, $L_{j-1,j-1}L_{j,j}^{-1} = A_j$ and the orthonormality of P_k implies $\sigma_{0,0} = I$ and $\sigma_{k,k}^{-1} = L_{k,k}^{-\mathrm{T}}K_{k,k}^{\mathrm{T}}$, with $K_{k,k}$ the leading coefficient of P_k. These equations will be used to derive expressions for the recursion coefficients in terms of the matrices $\sigma_{k,l}$.

The recursion coefficient D_0 is arbitrary and will be chosen equal to I. E_0 satisfies

$$E_0 = \langle xP_0, P_0\rangle = \sigma_{0,1}A_1^{\mathrm{T}} + B_0^{\mathrm{T}} = B_0 + A_1\sigma_{0,1}$$

and after some computation we find

$$D_k^{\mathrm{T}} = \langle xP_k, P_{k-1}\rangle = \sigma_{k,k}(L_{k-1,k-1}L_{k,k}^{-1})^{\mathrm{T}}\sigma_{k-1,k-1}^{-1} = \sigma_{k,k}A_k^{\mathrm{T}}\sigma_{k-1,k-1}^{-1},$$

$$E_k = \langle xP_k, P_k\rangle = \sigma_{k,k+1}A_{k+1}^{\mathrm{T}}\sigma_{k,k}^{-1} - \sigma_{k,k}A_{k+1}^{-\mathrm{T}}L_{k+1,k+1}^{-\mathrm{T}}L_{k+1,k}^{\mathrm{T}}A_{k+1}^{\mathrm{T}}\sigma_{k,k}^{-1} + K_{k,k}^{-\mathrm{T}}K_{k,k-1}^{\mathrm{T}}.$$

From the recurrence relations for $\{P_k\}$ and $\{Q_k\}$ we derive expressions for $K_{k,k}^{-\mathrm{T}}K_{k,k-1}^{\mathrm{T}}$ and $L_{k+1,k}L_{k+1,k+1}^{-1}$ which lead to $E_k = \sigma_{k,k+1}A_{k+1}^{\mathrm{T}}\sigma_{k,k}^{-1} - D_k^{\mathrm{T}}\sigma_{k-1,k}\sigma_{k,k}^{-1} + \sigma_{k,k}B_k^{\mathrm{T}}\sigma_{k,k}^{-1}$.

Thus we have the following algorithm:

(1) $\sigma_{-1,l} = 0$, $l = 1, 2, \ldots, 2n-2$,

(2) $\sigma_{0,l} = M_l$, $l = 0, 1, \ldots, 2n-1$,

(3) $D_0 = I$, $E_0 = \sigma_{0,1}A_1^{\mathrm{T}} + B_0^{\mathrm{T}}$.

(4) For $k = 1, 2, \ldots, n-1$ do

(4a) $D_kD_k^{\mathrm{T}} = (\sigma_{k-1,k+1}A_{k+1}^{\mathrm{T}} + \sigma_{k-1,k}B_k^{\mathrm{T}} - E_{k-1}\sigma_{k-1,k}$
$+ \sigma_{k-1,k-1}C_k^{\mathrm{T}} - D_{k-1}^{\mathrm{T}}\sigma_{k-2,k})A_k^{\mathrm{T}}\sigma_{k-1,k-1}^{-1}$

(4b) $\sigma_{k,k} = D_k^{-1}(\sigma_{k-1,k+1}A_{k+1}^{\mathrm{T}} + \sigma_{k-1,k}B_k^{\mathrm{T}} - E_{k-1}\sigma_{k-1,k} + \sigma_{k-1,k-1}C_k^{\mathrm{T}} - D_{k-1}^{\mathrm{T}}\sigma_{k-2,k})$

(4c) For $l = k+1, \ldots, 2n-k-1$ do
$\sigma_{k,l} = D_k^{-1}(\sigma_{k-1,l+1}A_{l+1}^{\mathrm{T}} + \sigma_{k-1,l}B_l^{\mathrm{T}} - E_{k-1}\sigma_{k-1,l} + \sigma_{k-1,l-1}C_l^{\mathrm{T}} - D_{k-1}^{\mathrm{T}}\sigma_{k-2,l})$

(4d) $E_k = \sigma_{k,k+1}A_{k+1}^{\mathrm{T}}\sigma_{k,k}^{-1} - D_k^{\mathrm{T}}\sigma_{k-1,k}\sigma_{k,k}^{-1} + \sigma_{k,k}B_k^{\mathrm{T}}\sigma_{k,k}^{-1}$

The algorithm is summarized in Fig. 1.

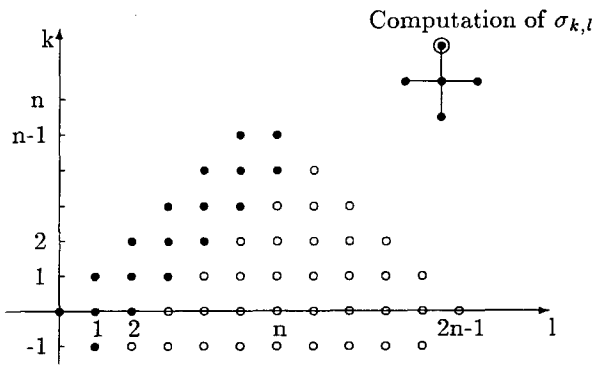

Fig. 1. Computation of $\sigma_{k,l}$: (●) matrix needed for the computation of D_k and E_k; and (○) matrix needed for the computation of other $\sigma_{k,l}$.

If we look at the algorithm, we see that the computations in loop (4c) can be performed simultaneously. The computation of the matrix $\sigma_{k,i}$ only depends on matrices of the two previous rows. To achieve a satisfactory loadbalance we proceed as follows: let r be the number of matrices to compute in the considered row, *nproc* the number of processors we are able to use and *npr* the number of processors we are using at the moment. Take $numb = \lceil r/npr \rceil$, $rst = r - (npr - 1) * numb$ and let the first processor compute *rst* elements and the other processors will each compute *numb* elements. This means that the first processor can have less $\sigma_{k,l}$'s to compute, but this will be compensated by the extra amount of work this processor has to perform in order to compute the recursion coefficients D_{k+1} and E_k.

To prevent under-utilization of processors, we introduce a parameter g and from the moment $numb \leqslant g$, we reduce the number of processors which will be used during the remaining part of the algorithm. We will keep processor 1 running and leave out the processor with the smallest number $\geqslant 2$. From the moment $npr = 1$, we put $numb = 0$ and $rst = r$.

After the computation of the matrices of the considered row, each processor has to send or receive elements from its neighbours. In order to know which processors are neighbours, each processor will store two variables *left* and *right* which indicate its neighbours. Besides these variables, each processor has at its disposal the numbers b_i and e_i, $i = 1, 2, 3$, which indicate which $\sigma_{k,*}$ were computed during the actual step, namely $\sigma_{k,b_1}, \ldots, \sigma_{k,e_1}$, and which will be computed during the two following steps, $\sigma_{k+1,b_2}, \ldots, \sigma_{k+1,e_2}$ and $\sigma_{k+2,b_3}, \ldots, \sigma_{k+2,e_3}$.

One can show that the number of operations of the serial algorithm is of the order $10p^3n^2$. If n is large enough the part of the parallel algorithm handled by $npr < nproc$ processors will be negligible. This means that the number of operations of the parallel algorithm is of the order $10p^3n^2/nproc$.

Of course, the communication cannot be neglected. We can show that the number of messages is given by $N_{\text{mess}} \approx (2nproc - 1 - 2/nproc)n$, while the total length of the messages N_{len}, expressed in bytes, satisfies

nproc	2	3	4	$\geqslant 5$
$N_{\text{len}} \approx$	$72p^2n$	$\approx (416/3)p^2n$	$\approx 212p^2n$	$\approx 80nproc\,p^2n$

The ratio communication/computation, but also the fact that the number of messages increases as the number of processors increases, while the number of operations decreases, show that n and p have to be large enough to achieve satisfactory speed-up.

An implementation of the serial and parallel algorithm on a SP1, using the message passing library PVM, leads to the following results (Tables 1 and 2, Figs. 2 and 3).

Example 1. Let $p = 2$ and take $W(x) = 0.5I$ on $[-1, 1]$. Then the modified moments with respect to the diagonal matrix polynomials with the Chebyshev polynomials of the 2nd kind on the diagonal ($A_n = C_n = 0.5I$ and $B_n = 0, n = 0, 1, \ldots$) are given by $M_{2l} = [1/(2l + 1)]I, M_{2l+1} = 0, l = 0, 1, 2, \ldots$.

First we determine the optimal value of the parameter g which depends on the machine we are working with, the number of processors and the block size of the matrices. The execution times on 4 processors for $g = 20, 40, 60, 80$ are very close, but for $g = 40$ they are the smallest. Using $g = 40$. even for a different number of processors, leads to the execution times (in s) shown in Table 1.

Table 1

n	200	400	600	800	1000	1200	1400	1600	1800	2000
Serial	2.630	10.428	23.739	41.911	67.178	94.404	128.551	167.432	211.944	261.646
$nproc = 2$	2.336	6.741	14.029	23.863	36.459	51.550	69.539	89.744	112.687	138.472
Speed-up	1.13	1.55	1.69	1.76	1.84	1.83	1.85	1.87	1.88	1.89
$nproc = 4$	2.144	5.656	10.601	16.870	24.485	33.515	43.776	55.408	68.228	82.619
Speed-up	1.23	1.84	2.24	2.48	2.74	2.82	2.94	3.02	3.11	3.17
$nproc = 8$	2.744	5.791	10.916	16.898	22.003	29.371	37.433	44.867	53.115	62.546
Speed-up	0.96	1.80	2.17	2.48	3.05	3.21	3.43	3.73	3.99	4.18

Table 2

n	200	400	600	800	1000	1200	1400	1600	1800	2000
Serial	5.066	20.238	45.522	80.976	126.525	182.174	247.855	323.592	409.747	509.383
$nproc = 2$	3.907	12.284	26.312	45.771	69.354	99.071	133.093	175.172	217.823	271.258
Speed-up	1.30	1.65	1.73	1.77	1.82	1.84	1.86	1.85	1.88	1.88
$nproc = 4$	3.573	9.709	18.478	30.060	44.281	61.333	80.707	102.833	127.902	155.498
Speed-up	1.42	2.08	2.46	2.69	2.86	2.97	3.07	3.15	3.20	3.28
$nproc = 8$	4.434	10.267	18.042	25.908	36.261	47.894	60.414	74.133	89.522	106.547
Speed-up	1.14	1.97	2.52	3.13	3.49	3.80	4.10	4.37	4.58	4.78

Example 2. Let $p = 5$ and take

$$W(x) = \begin{pmatrix} \frac{1}{2} & \frac{1}{8}x & 0 & 0 & 0 \\ \frac{1}{8}x & \frac{1}{2} & \frac{1}{4}x & 0 & 0 \\ 0 & \frac{1}{4}x & \frac{1}{2} & \frac{1}{2\sqrt{2}}x & 0 \\ 0 & 0 & \frac{1}{2\sqrt{2}}x & \frac{1}{2} & \frac{1}{2\sqrt{10}}x \\ 0 & 0 & 0 & \frac{1}{2\sqrt{10}}x & \frac{1}{2} \end{pmatrix}$$

on the interval $[-1, 1]$. Then the modified moments with respect to the diagonal matrix polynomials with the Chebyshev polynomials of the 2nd kind on the diagonal ($A_n = C_n = 0.5I$ and $B_n = 0, n = 0, 1, \ldots$) are given by $M_{2l} = [1/(2l + 1)]I$ and M_{2l+1} is a tridiagonal matrix with zeros on the main diagonal and, on both subdiagonals, the vector $(a/4, a/2, a/\sqrt{2}, a/\sqrt{10})$ where $a = 2(l + 1)/[(2l + 1)(2l + 3)]$.

Again the execution times of the algorithm on 4 processors with $g = 20, 40, 60$ are very close. If we take $g = 40$, we get the execution times (in s) as shown in Table 2.

The speed-ups on 2 and 4 processors are very close to the maximum speed-up of 2 and 4, respectively. In the case where we are using 8 processors, the time spent in communication becomes

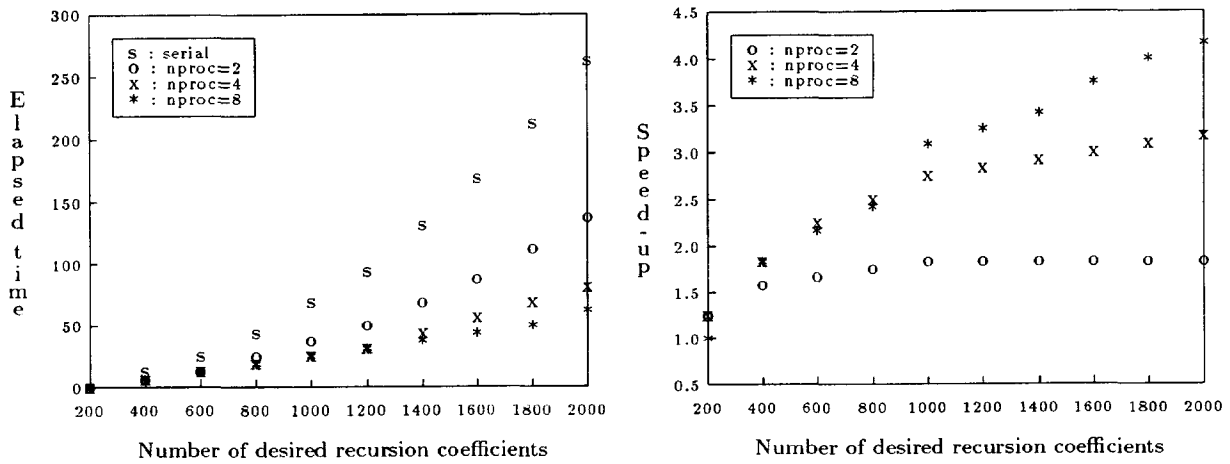

Fig. 2.

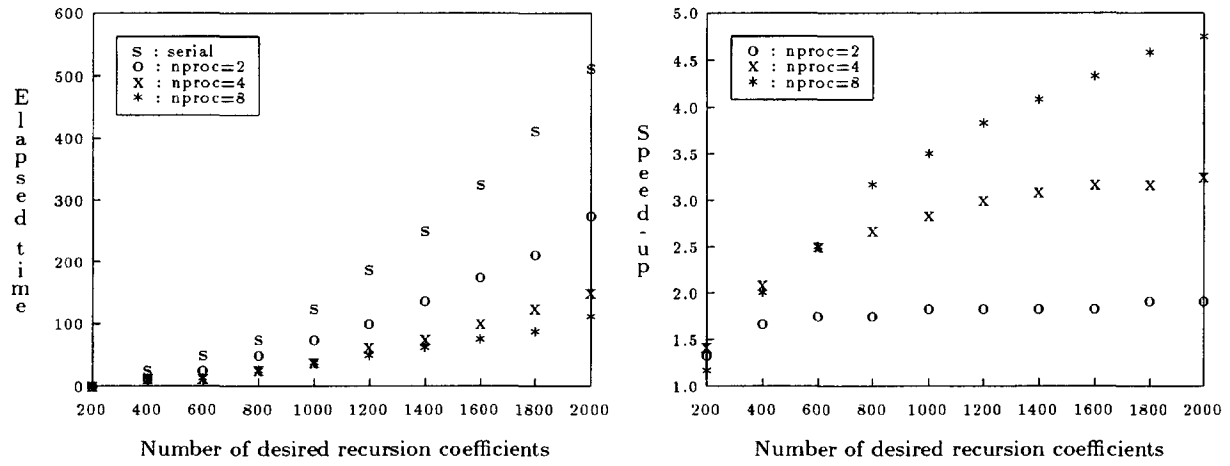

Fig. 3.

significant in comparison with the time spent in computation and this will influence the speed-up. For larger n the results improve since the time spent in computation increases much faster than the time spent in communication.

4.2. Gaussian quadrature

As already mentioned, the Gaussian quadrature weights can be computed by means of the eigenvalues and the first p components of the orthonormalized eigenvectors of a symmetric band-matrix with semi-bandwidth p. So the construction of a parallel algorithm for the approximation

of a matrix integral will be mainly the construction of a parallel algorithm for the computation of the eigensystem of a symmetric bandmatrix.

The classical approach for this problem consists of two steps (see, for example, the routines of the LAPACK-library), first the given matrix A is reduced to a similar tridiagonal matrix T and then the tridiagonal eigenproblem is solved by means of the QR or QL algorithm.

Going through the literature of algorithms for the solution of eigenproblems on multiprocessors we find parallel algorithms to reduce the given matrix to a similar tridiagonal matrix, e.g. [12, 19, 20, 22, 25]. The eigensystem of the symmetric tridiagonal matrix can be determined by the parallel QR-algorithm, see e.g. [3, 4, 23, 24]. On the other hand, we often find divide-and-conquer methods, [1, 2, 8, 10, 11]. In [1] a divide and conquer method to compute the eigensystem of a symmetric bandmatrix is described. The algorithm was implemented on a shared memory multi-processor. We will adapt this algorithm in order to compute the quadrature weights and implement it on a SP.

Suppose $A = (a_{i,j})_{i,j=1,N}$ is a symmetric bandmatrix with semi-bandwidth p ($a_{i,j} = 0$ for $|i-j| > p$). Choose k_1 and k_2 such that $k_1 + k_2 + p = N$ and let

$$A = \begin{pmatrix} A_1 & V_1 & 0 \\ V_1^{\mathrm{T}} & \Delta & V_2^{\mathrm{T}} \\ 0 & V_2 & A_2 \end{pmatrix}, \quad A_i \in \mathbb{R}^{k_i \times k_i},\ V_i \in \mathbb{R}^{k_i \times p},\ \Delta \in \mathbb{R}^{p \times p}.$$

Perform a similarity transformation of A:

$$A = P^{\mathrm{T}}AP = \begin{pmatrix} I_{k_1} & 0 & 0 \\ 0 & 0 & I_{k_2} \\ 0 & I_p & 0 \end{pmatrix} \begin{pmatrix} A_1 & V_1 & 0 \\ V_1^{\mathrm{T}} & \Delta & V_2^{\mathrm{T}} \\ 0 & V_2 & A_2 \end{pmatrix} \begin{pmatrix} I_{k_1} & 0 & 0 \\ 0 & 0 & I_p \\ 0 & I_{k_2} & 0 \end{pmatrix} = \begin{pmatrix} A_1 & 0 & V_1 \\ 0 & A_2 & V_2 \\ V_1^{\mathrm{T}} & V_2^{\mathrm{T}} & \Delta \end{pmatrix}$$

and define

$$V = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}, \qquad A_0 = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}.$$

The spectral decompositions

$$A_i = Q_i \Delta_i Q_i^{\mathrm{T}}, \qquad \Delta_i = \operatorname{diag}(\lambda_0^{(i)} \ldots \ \ldots \lambda_{k_i-1}^{(i)}), \quad i = 1, 2$$

can be computed by means of the classical algorithms or we can perform the divide and conquer algorithm to further reduce the order of the blocks. Once the spectral decompositions of the submatrices are known the spectral decomposition of A can be computed by means of the following algorithm:

(1) Combine and sort the eigenvalues of A_1 and A_2: $\lambda_0^{(0)} \leqslant \lambda_1^{(0)} \leqslant \cdots \leqslant \lambda_{N-p-1}^{(0)}$.
(2) Let $\hat{\lambda}_0^{(0)} < \hat{\lambda}_1^{(0)} < \cdots < \hat{\lambda}_{m-1}^{(0)}$, be the m distinct eigenvalues of $\sigma(A_0)$.
(3) Determine $\hat{\lambda}_{-1}^{(0)}$ and $\hat{\lambda}_m^{(0)}$ such that $\hat{\lambda}_{-1}^{(0)} \leqslant \lambda_j \leqslant \hat{\lambda}_m^{(0)}$, $j = 0, 1, \ldots, N-1$, where λ_j are the eigenvalues of A.
(4) Determine $U = \begin{pmatrix} Q_1^{\mathrm{T}} & 0 \\ 0 & Q_2^{\mathrm{T}} \end{pmatrix} V$.
(5) For $j = -1, 0, \ldots, m$ do

(a) Determine the extended Weinstein matrix

$$We(\hat{\lambda}_j^{(0)}) = \begin{pmatrix} \Delta - \hat{\lambda}_j^{(0)} I_p - V^{\mathrm{T}}(A_0 - \hat{\lambda}_j^{(0)} I)^{+} V & V^{\mathrm{T}} Q \\ Q^{\mathrm{T}} V & 0_\mu \end{pmatrix},$$

and compute its inertia (neg_j, nul_j, pos_j). (0_μ is the $\mu \times \mu$ matrix with all components equal to 0 and B^+ is the pseudo-inverse of B).

(b) If $We\hat{\lambda}_j^{(0)})$ is singular,

- Determine the null space $\mathcal{N}(We(\hat{\lambda}_j^{(0)}))$.
- Determine the corresponding orthonormal basis of the eigenspace of A.

(6) For $j = 0, 1, \ldots, m$ (determine the eigenvalues in the interval $(\hat{\lambda}_{j-1}^{(0)}, \hat{\lambda}_j^{(0)})$).

(a) Let M be equal to the number of eigenvalues of A in the interval $(\hat{\lambda}_{j-1}^{(0)}, \hat{\lambda}_j^{(0)})$.

(b) If $M > 0$

- Isolate the eigenvalues in the interval by means of bisection, using the inertia of the Weinstein matrix $W(\lambda) = \Delta - \lambda I_p - V^{\mathrm{T}}(A_0 - \lambda I)^{-1} V$.
- Determine the isolated eigenvalues by means of a 'zerofinder' applied on $\det W(\lambda) = 0$.
- Determine the corresponding eigenvectors.

Let us discuss some points of this algorithm. To determine $\hat{\lambda}_{-1}^{(0)}$ and $\hat{\lambda}_m^{(0)}$ such that $\hat{\lambda}_{-1}^{(0)} \leqslant \lambda_j \leqslant \hat{\lambda}_m^{(0)}$, $j = 0, 1, \ldots, N-1$, we can use Gerschgorin's circle theorem or the property $|\lambda_j| \leqslant \|A\|$. During the last "put together" step, when computing the eigenvalues of the given matrix, we can use the property that all zeros of an orthonormal matrix polynomial lie in the interval of orthogonality.

The inertia of symmetric matrices can be determined by means of the algorithm described in [6, 7, 9] and which is based on the LDL^{T} decomposition, where L is a lower triangular matrix with 1 on the main diagonal and D is a block diagonal matrix with blocks of order 1 or 2.

From the moment an eigenvalue is isolated we will use a "zerofinder" to determine the zero of $\det W$. We could continue with the bisection method but we prefer to use an algorithm which is faster in many cases. The algorithm uses bisection, linear and inverse quadratic interpolation, and was published by Dekker in 1969. In 1973 Brent published an improved version (see [5]). The Fortran algorithm of this version can be found in [13].

To determine the eigenvectors corresponding to the eigenvalue $\hat{\lambda}_j^{(0)}$ of A_1 or A_2, but which also belongs to the spectrum of A, we determine a basis $\{\binom{x_i}{y_i}\}$ of the null space $\mathcal{N}(We(\hat{\lambda}_j^{(0)}))$ and orthonormalize these vectors with respect to the inner product

$$\left\langle \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right\rangle = (x_i^{\mathrm{T}} y_i^{\mathrm{T}}) \begin{pmatrix} I + V^{\mathrm{T}}(\lambda I - A_0)^{+^2} V & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} x_j \\ y_j \end{pmatrix}.$$

The transformed vectors

$$z_i = \begin{pmatrix} (\lambda I - A_0)^{+} V x_i + Q y_i \\ x_i \end{pmatrix}$$

form an orthonormal basis of the eigenspace of $\bar{A}$ corresponding to the eigenvalue $\hat{\lambda}_j^{(0)}$ and with respect to the classical inner product $z_i^{\mathrm{T}} z_j = \delta_{i,j}$ and the vectors $\{Pz_i\}$ form an orthonormal basis of the eigenspace of A corresponding to the eigenvalue $\hat{\lambda}_j^{(0)}$.

If $\lambda \in (\hat{\lambda}_{j-1}^{(0)}, \hat{\lambda}_j^{(0)})$ is an eigenvalue of A, we determine a basis $\{v_i\}$ of the null space $\mathcal{N}(W(\lambda))$ and orthonormalize these vectors with respect to the inner product

$$\langle v_i, v_j \rangle = v_i^{\mathrm{T}}(I + V^{\mathrm{T}}(A_0 - \lambda I_{N-p})^{-2}V)v_j.$$

The images

$$z_i = \begin{pmatrix} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \begin{pmatrix} \lambda I - \Delta_1 & 0 \\ 0 & \lambda I - \Delta_2 \end{pmatrix}^{-1} U v_i \\ v_i \end{pmatrix}$$

form an orthonormal basis of the eigenspace of $\bar{A}$ corresponding to λ and the vectors $\{Pz_i\}$ form an orthonormal basis of the eigenspace of A corresponding to the eigenvalue λ.

In contrast to the method described in [1], where first the vectors are transformed and then they are orthonormalized with respect to the classical inner product, this method has the advantage that we do not have to perform the complete transformation of the vectors if we are only interested in the first p components of the normalized eigenvectors.

Let us now describe the implementation of the algorithm on a distributed memory multiprocessor. Suppose the number of processors is equal to a power of 2, $nproc = 2^c$. Then the given matrix of order $N = np$ is divided into $nproc$ submatrices. For example, if $nproc = 2^3$ we divide the given system as shown in Fig. 4.

Since each pair of submatrices is separated by a band of order p, the total order of the subsystems will be given by $np - (1 + 2 + 4 + \cdots + 2^{c-1})p = np - (2^c - 1)p$. This implies that the order of the first $nproc - 1$ submatrices is equal to $\lceil [np - (2^c - 1)p]/nproc \rceil$, while the last submatrix is of order $np - (2^c - 1)p - (nproc - 1)\lceil [np - (2^c - 1)p]/nproc \rceil$.

The eigensystems of these submatrices are determined by means of the LAPACK-routine for bandsymmetric matrices. This can be done simultaneously for the different submatrices.

Afterwards these eigensystems are assembled. At each step, neighbouring processors need to communicate. To reduce the communication time we will put the eigensystem of the larger matrix in the memory of one of the processors which was involved in the computation of the smaller eigensystems. For example, if $nproc = 2^3$ we proceed as shown in Fig. 5.

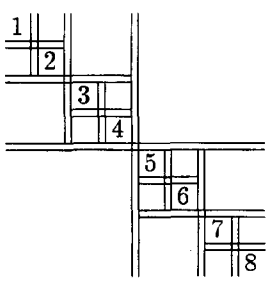

Fig. 4.

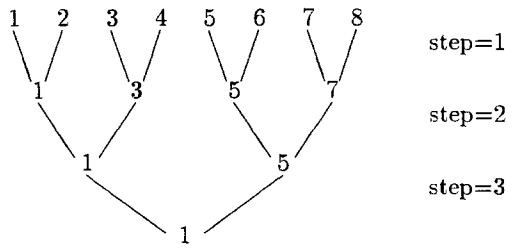

Fig. 5.

In step 1, processors 1 and 2, 3 and 4, 5 and 6, 7 and 8 combine the eigensystems they have computed to get the eigensystem of the larger matrix. The results are put in processor 1, 3, 5 and 7, respectively. During step 2, processors 1 and 3, 5 and 7 combine their systems and put the solution in processors 1 and 5, respectively. During the third step the two remaining processors combine their solution but instead of computing the normalized eigenvectors, they compute the first p components of the normalized eigenvectors and determine the partial approximation of the given integral.

In general processor $2^i m + 1$ exchanges information with processor $2^{i-1}(2m + 1) + 1$, $m = 0, 1, \ldots, 2^{c-i} - 1$, during step i, $1 \leqslant i \leqslant c$.

The host program will take care of the distribution of the data among the different processors. In the beginning of the program the host will send the submatrices A_i to the different processors and during the execution the host will send the matrices V and Δ to the corresponding processors.

Also in the "put together" steps some computations can be performed simultaneously. Looking at the algorithm, we notice U can be computed in parallel and the computations of loops (5) and (6) can be treated simultaneously. Since two processors are working together, each of them will treat $m/2$ eigenvalues $\hat{\lambda}_j^{(0)}$ and decide whether these values are eigenvalues of the larger matrix. Afterwards, each of the processors will treat half of the intervals to look for the eigenvalues in the intervals.

Let us have a look at the complexity of the different programs. In the serial algorithm, we distinguish two parts. First we determine the eigenvalues and eigenvectors by means of the LAPACK-routine. The number of operations for this part is of the order $N^2([(15p - 6)/p]N + 12p - 21)$ (see [1]).

Afterwards we determine the quadrature weights. Suppose we have $N = np$ different zeros. Then the computation of a quadrature weight can be done in $2p^2$ operations. If F and G are matrix polynomials of degree $gr\,f$ and $gr\,g$, respectively, we get the function evaluations after performing $2p^2(gr\,f + gr\,g)$ operations. Of course F and G can be other matrix functions. To get the new partial approximation, 2 matrix products and 1 sum have to be performed. So the total number of operations for this second part is given by $N(4p^3 + 3p^2 + 2p^2(gr\,f + gr\,g))$ and the total number of operations for the serial algorithm satisfies $N_{\text{ser}} \approx 15N^3$.

To get an idea of the number of operations for the parallel performance of the divide and conquer method, we suppose we have $nproc = 2^q$ processors with q the height of the binary tree. We do not take into account the order of the extended Weinstein matrices $We(\cdot)$ and we suppose we need s evaluations of a Weinstein matrix in order to locate an eigenvalue. Further we suppose the

length of the different subsystems of the same step is equal to $N_{\text{step}} = N/2^{(c-\text{step})}$. The number of operations is given by

$$N_{\text{par}} \approx \left(\frac{4}{7}\,nproc^3 + \frac{101}{7}\right)\frac{N^3}{nproc^3} + (2s(p^2+2p+2) + 2p^2 + 5p)\frac{N^2}{3}.$$

This leads to the following upper bounds for the speed-up of the parallel divide and conquer method with respect to the serial algorithm using the LAPACK-routine

$$S_{nproc} = \frac{N_{\text{ser}}}{N_{\text{par}}} \approx \frac{15N^3\,nproc^3}{(\frac{4}{7}nproc^3 + \frac{101}{7})N^3} = \frac{105\,nproc^3}{4\,nproc^3 + 101}.$$

Of course these are upper bounds since we only take into account the terms of N^3. Moreover, we do not take into account the communication. In the expression for the communication we do not consider the time spent in the sending of the subsystems to the different nodes. If you look at the parallel modified Chebyshev algorithm to compute the recursion coefficients, we see that each processor knows these coefficients and thus it will not be necessary to send the subsystems to the different nodes.

We can show that the number of messages satisfies $N_{\text{mess}} = 5nproc - 4$ and the total length of the messages, expressed in bytes, is given by

$$N_{\text{len}} = N^2 8\,\frac{nproc-2}{nproc} + 4N((2p+3)(q-1) + 3p + 2) + 16p(p+1)(nproc-1) + 16p2.$$

The ratio

$$\frac{N_{\text{len}}}{N_{\text{par}}} = \frac{56(nproc-2)nproc^2}{(4nproc^3 + 101)np}$$

shows that n and p have to be large enough to achieve satisfactory speed-up.

Example 3. Let $p = 5$ and let the matrix density function on the interval $[-1, 1]$ be given by

$$W(x) = \begin{pmatrix} \frac{1}{2} & \frac{1}{8}x & 0 & 0 & 0 \\ \frac{1}{8}x & \frac{1}{2} & \frac{1}{4}x & 0 & 0 \\ 0 & \frac{1}{4}x & \frac{1}{2} & \frac{1}{2\sqrt{2}}x & 0 \\ 0 & 0 & \frac{1}{2\sqrt{2}}x & \frac{1}{2} & \frac{1}{2\sqrt{10}}x \\ 0 & 0 & 0 & \frac{1}{2\sqrt{10}}x & \frac{1}{2} \end{pmatrix}.$$

Let F be the matrix polynomial of degree 30 and G the matrix polynomial of degree 20. To get an idea of the errors we have computed the integral

$$\int_a^a F(x)W(x)G(x)^{\mathrm{T}}\,\mathrm{d}x$$

Table 3

n	Serial	$nproc = 2$	$nproc = 4$	$nproc = 8$
50	0.13699E − 13	0.43518E − 14	0.11923E − 13	0.14695E − 13
100	0.24433E − 13	0.26206E − 12	0.12839E − 12	0.95232E − 14
150	0.90158E − 14	0.21837E − 13	0.76537E − 13	0.22917E − 01
200	0.58915E − 13	0.19486E − 12	0.10690E − 12	0.45382E − 03
250	0.11773E − 12	0.32217E − 12	0.13992E − 12	0.17665E − 02
300	0.11966E − 12	0.37644E − 13	0.54466E − 11	0.85718E − 01

Table 4

n	50	100	150	200	250	300
Serial	4.393	31.297	104.634	236.321	468.341	781.134
$nproc = 2$	4.740	21.045	51.868	100.217	169.953	260.569
Speed-up	0.79	1.49	2.02	2.36	2.76	3.00
$nproc = 4$	5.605	22.244	56.467	109.911	185.709	281.707
Speed-up	0.78	1.41	1.85	2.15	2.52	2.77
$nproc = 8$	6.988	22.865	56.371	110.634	183.944	285.843
Speed-up	0.63	1.37	1.86	2.14	2.55	2.73

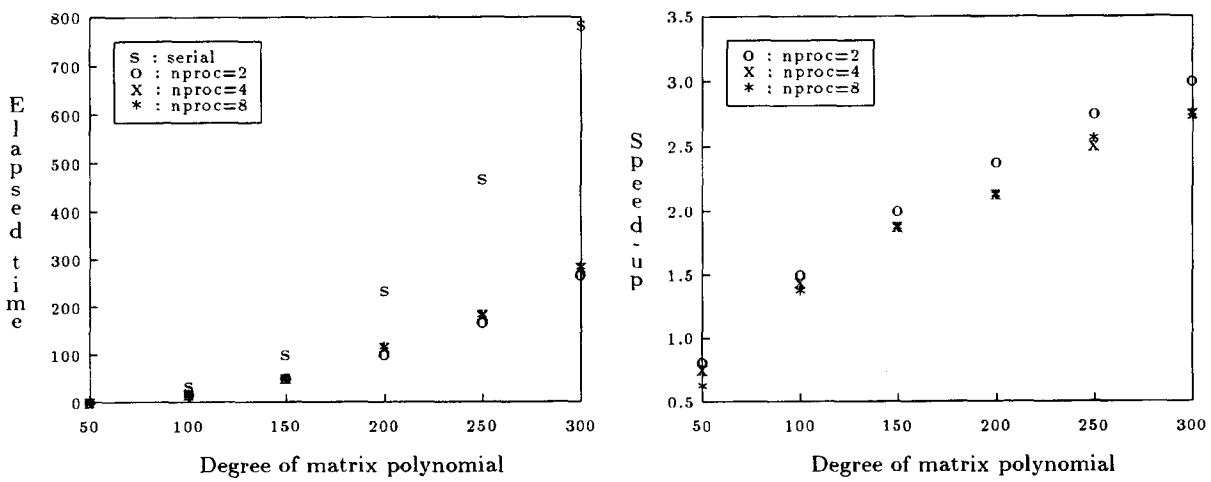

Fig. 6.

by means of Mathematica with a precision of 30 digits and compared these results with the results of the implementation of the Gaussian quadrature rules. Notice the quadrature formules are, theoretically, correct for $n \geqslant 26$, but since we are interested in the timings and relative errors, we perform the algorithm for bigger n.

The relative errors are given in Table 3.

When using 8 processors, the large errors are due to errors in the computation of determinants to compute the eigenvalues of a matrix out of the solution of two smaller problems. It would be better to compute these determinants in higher precision. Another source of errors is the orthonormalization of the eigenvectors. These round-off errors lead to the fact that, during some steps, the algorithm does not compute enough eigenvalues and eigenvectors and this leads to the large, unacceptable relative errors.

The execution times (in s) and speed-ups using a SP1 are given in Table 4; see also Fig. 6.

Performing some analysis about the time spent in communication and computation will give an explanation for the poor results when using 8 processors. The number of operations performed in 1 second (Mflop/s) is equal to 70, 49, 17, 13 for 1, 2, 4 and 8 processors, respectively. This very big decrease of operations performed in 1 s is due to the fact that we have only counted the number of operations (sum, subtraction, product, division) and did not take into account the large number of tests which require much more time than an operation. This is also the reason why the results are poor for a large number of processors. In this cases the "put together" steps require much more time due to the large number of tests which have to be performed and cannot balance the larger number of operations performed by the LAPACK-routine when using less processors.

Using 2 processors gives very good results and in this case also the relative errors are acceptable.

Acknowledgements

I would like to thank W. Van Assche for his careful reading of this manuscript.

References

[1] P. Arbenz, Divide and conquer algorithms for the bandsymmetric eigenvalue problem, *Parallel Comput.* **18** (1992) 1105–1128.

[2] P. Arbenz and G.H. Golub, On the spectral decomposition of hermitian matrices modified by low rank perturbations with applications, *SIAM J. Matrix Anal. Appl.* **9** (1988) 40–58.

[3] Z. Bai and J. Demmel, On a block implementation of Hessenberg multishift QR iteration, *Int. J. High Speed Comput.* **1** (1989) 97–112.

[4] C.H. Bischof, A parallel QR factorization algorithm with controlled local pivoting, *SIAM J. Sci. Statist. Comput.* **12** (1991) 36–57.

[5] R.P. Brent, *Algorithms for Minimization Without Derivatives* (Prentice-Hall, Englewood Cliffs, NJ, 1973).

[6] J.R. Bunch, Analysis of the diagonal pivoting method, *SIAM J. Numer. Anal.* **8** (1971) 656–680.

[7] J.R. Bunch and L. Kaufman, Some stable methods for calculating inertia and solving symmetric linear systems, *Math. Comput.* **31** (1977) 163–179.

[8] J.R. Bunch, C.P. Nielsen and D.C. Sorensen, Rank-one modification of the symmetric eigenproblem, *Numer. Math.* **31** (1978) 31–48.

[9] J.R. Bunch and B.N. Parlett, Direct methods for solving symmetric indefinite systems of linear equations, *SIAM J. Numer. Anal.* **8** (1971) 639–655.

[10] J.J.M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numer. Math.* **36** (1981) 177–195.

[11] J.J. Dongarra and D.C. Sorensen, A fully parallel algorithm for the symmetric eigenvalue problem, *SIAM J. Sci. Statist. Comput.* **8** (1987) 139–154.

[12] J.J. Dongarra and R.A. van de Geijn, Reduction to condensed form for the eigenvalue problem on distributed memory architectures, *Parallel Comput.* **18** (1992) 973–982.
[13] G.E. Forsythe, M.A. Malcolm and C.B. Moler, *Computer Methods for Mathematical Computations* (Prentice-Hall, Englewood Cliffs, NJ, 1977).
[14] W. Gautschi, On generating orthogonal polynomials, *SIAM J. Sci. Statist. Comput.* **3** (1982) 289–317.
[15] W. Gautschi, Orthogonal polynomials: constructive theory and applications, *J. Comput. Appl. Math.* **12** (1985) 61–76.
[16] W. Gautschi, Computational aspects of orthogonal polynomials, in: P. Nevai, Ed., *Orthogonal Polynomials: Theory and Practice*, Vol. 294, NATO ASI Series C (Kluwer, Dordrecht, 1990) 181–216.
[17] J.S. Geronimo, Scattering theory and matrix orthogonal polynomials on the real line, *Circuits Systems Signal Process* **1** (1982) 471–495.
[18] I. Gohberg, P. Lancaster and L. Rodman, *Matrix Polynomials* (Academic Press, New York, 1982).
[19] T.Z. Kalamboukis, A parallel algorithm for the dense symmetric eigenvalue problem on a transputer array, *Parallel Comput.* **18** (1992) 207–212.
[20] L. Kaufman, The generalized Householder transformation and sparse matrices, *Linear Algebra Appl.* **90** (1987) 221–234.
[21] M. Krein, Fundamental aspects of the representation theory of Hermitian operators with deficiency index (m,m), *Ukrain. Mat. Z.* **1** (1949), 3–66; English translation in *Amer. Math. Soc. Transl.* **97** (1970) 75–143.
[22] B. Lang, A parallel algorithm for reducing symmetric banded matrices to tridiagonal form, *SIAM J. Sci. Statist. Comput.* **14** (1993) 1320–1338.
[23] G.S. Miminis and C.C. Paige, Implicit shifting in the QR and related algorithms, *SIAM J. Matrix Anal. Appl.* **12** (1991) 385–400.
[24] Th. Schreiber, P. Otto and F. Hofmann, A new efficient parallelization strategy for the QR algorithm, *Parallel Comput.* **20** (1994) 63–75.
[25] R. Schreiber and B. Parlett, Block reflectors: theory and computation, *SIAM J. Numer. Anal.* **25** (1988) 189–205.
[26] A. Sinap and W. Van Assche, Polynomial interpolation and Gaussian quadrature for matrix-valued functions, *Linear Algebra Appl.* **207** (1994) 71–114.
[27] A. Sinap and W. Van Assche, Orthogonal matrix polynomials and applications, *J. Comput. Appl. Math.* **66** (1996), to appear.